2024-2 시계열자료분석팀 클린업 3주차

소속	성균관대학교 통계분석학회 P-SAT
작성자	33기 김나현

목차

I. 시계열 데이터의 전처리

1. 결측치 보간

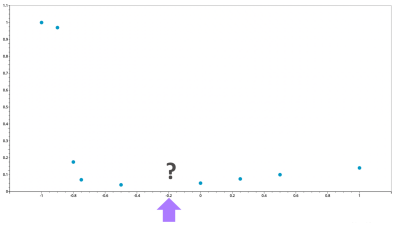
데이터 분석을 진행하다보면 다수의 결측치를 발견할 수 있고, 이때 안정적인 분석을 위해 결측치를 보간해주어야 합니다. 하지만 일반적인 자료에서 사용하는 평균, 최빈값 등의 방법을 통해 결측치를 보간할 경우, 시계열 데이터의 특성을 반영하지 못해서 해당 시점의 평균, 분산에 왜곡이 생기는 등의 문제가 발생할 수 있습니다. 따라서 시계열 데이터는 아래의 방법을 통해 결측치를 보간합니다.

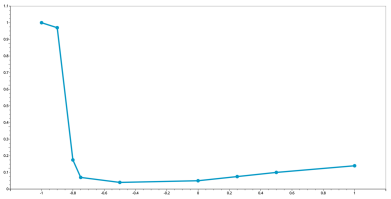
> 🥦 **시계열 데이터의 결측치 보간법**
>
> i. **LOCF**(Last Observation Carried Forward): 직전 관측치 값으로 결측치 대체
>
> ii. **NOCB**(Next Observation Carried Backward): 직후 관측치 값으로 결측치를 대체
>
> iii. **Moving Average / Moving Median** : 직전 N의 time window의 평균치 / 중앙값으로 대체

일반적으로는 위 3가지 방법을 사용하지만, 결측치를 기준으로 패턴이 급격하게 변화하는 경우에는 적합하지 않을 수 있습니다. 예를 들어, 주가를 예측하는 경우 상승 구간이거나 하락 구간에 결측치가 존재할 때 이전 값들을 이용하여 대체하게 되면 실제 값과 차이가 꽤 발생하게 되겠죠? 따라서, 이러한 경우들에서는 아래의 방법들을 고려할 수 있습니다.

iv. **선형 / 비선형 보간법**

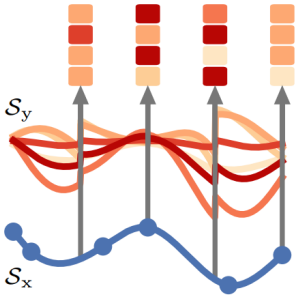
결측치를 중심으로 window를 설정하고 해당 구간 내에 선형 또는 다항 함수를 적합하여 적합된 함수를 통해 결측치를 보간합니다.





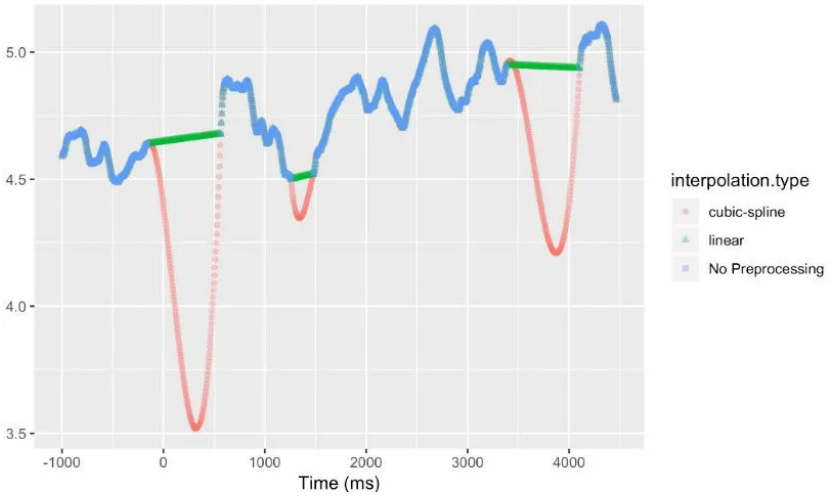
v. **스플라인 보간법**

결측치를 중심으로 한 window의 전체 값을 한 번에 적합하지 않고, 이를 소구간으로 분할하여 스플라인(각 구간마다 함수를 적합하고 모든 구간에서 함수가 매끄럽게 이어지도록 하여 전체 구간에 함수를 적합하는 방법)을 적용하여 결측치를 보간합니다.



vi. **모델링을 통한 결측값 예측**

(결측치가 너무 많은 경우) 각 결측치마다 이전까지의 시계열 데이터를 활용해 모델링한 다음, 각 결측값을 예측하여 보간합니다.



2. 노이즈 처리

노이즈란 의도하지 않은 데이터의 왜곡을 불러오는 모든 것을 의미합니다. 예를 들어, 평균적인 대중교통 이용량에 대해 분석하고자 할 때, 유명 가수의 콘서트로 인해 이용량이 급격하게 상승한 것은 연구자의 관점에 따라 노이즈로 볼 수 있습니다. 이런 경우에 노이즈를 제거할 수 있는 denoising 기법에 대해 간단히 알아보겠습니다.

(1) 평활 (Smoothing)

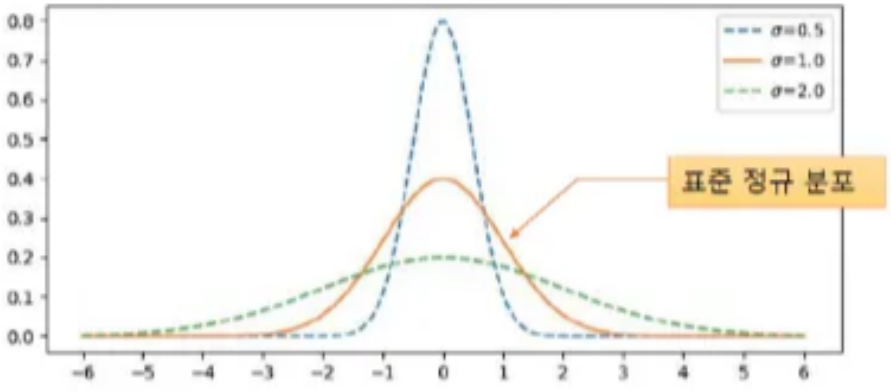
이동평균평활법(MA) 또는 지수평활법을 통해 denoising 할 수 있습니다. 이 방법은 노이즈가 많이 발생하는 데이터에서는 노이즈 자체가 평균에 반영되기 때문에 적절하지 않고, 노이즈가 적은 데이터에 효과적입니다. (노이즈가 평균값에 반영되므로, 그 수가 많은 경우 평균 역시 노이즈의 성격을 띨 수 있기 때문에!) 금융 데이터 노이즈 제거에 자주 사용됩니다. 이동평균평활법과 지수 평활 모두 이미 다룬 개념이기 때문에 넘어가도록 하겠습니다!

(2) Filtering

노이즈가 많은 환경에서는 노이즈를 Filtering할 수 있습니다. 현재 시계열에 발생하는 노이즈가 어떤 특정한 분포를 따른다고 가정하고 해당 분포의 값을 시계열에서 제거하는 형태로 진행합니다.

i. **가우시안 필터링(Gaussian Filtering)**

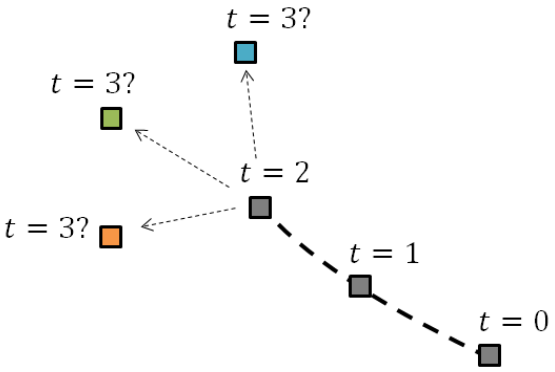
$$G(x,y) = \frac{1}{2\pi\sigma^2} exp(-\frac{x^2+y^2}{2\sigma^2})$$



가우시안 필터는 노이즈가 정규분포를 따른다고 가정한 뒤, 중심에 가까운 데이터에는 더 큰 가중치를, 멀어질수록 작은 가중치를 부여합니다. 가우시안 필터도 평활 방법의 일종으로 볼 수 있으며, 주로 이미지 처리에서 블러링 또는 노이즈 처리에 사용되지만 시계열 데이터에도 적용 가능합니다.

ii. **칼만 필터 (Kalman Filter)**

우선, 칼만 필터를 어떠한 경우에 사용할 수 있는지 개념적으로 이해해보도록 하겠습니다.



그림에서 볼 수 있듯이 지금까지 t = 0, 1, 2이라는 시간 순서에 따라 궤적을 얻었다고 해보겠습니다. 그러면 t = 3 일 때는 물체가 어디에 있다고 보는 것이 가장 타당할까요? 아마도 초록색 네모가 가장 타당한 다음 위치라고 생각할 수 있을 것입니다. 이처럼 칼만필터를 이용하면 물체를 추적 할 때 지금까지의 궤적에 기반해 다음번 물체의 위치를 추정하는데 사용할 수 있다.

즉, Kalman filter는 동적 시스템(dynamic system)의 현 시점의 true값을 추정할 수 있습니다. 예를 들어, GPS를 단 자전거를 타고 있는 친구의 위치를 찾고 싶다고 하겠습니다. GPS가 있음에도 그 정확도가 떨어져 가끔 잘못된 위치 추정값을 줍니다. 친구의 진짜 위치 (true값)을 알기 위해 측정의 불확실성과 노이즈를 고려하여 가장 가능성 있는 상태를 찾는 방법이 kalman filter입니다.

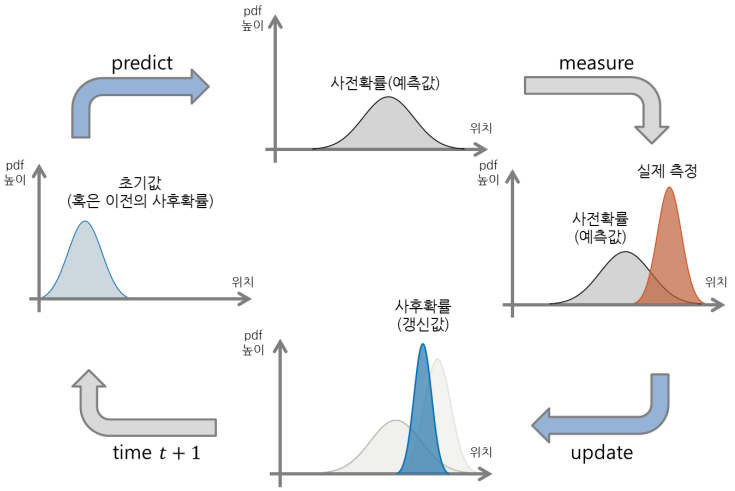
Kalman filter는 예측 단계와 업데이트 단계로 구성되어 있습니다. 두 단계를 개념적으로 이해해보겠습니다.

- 예측 단계

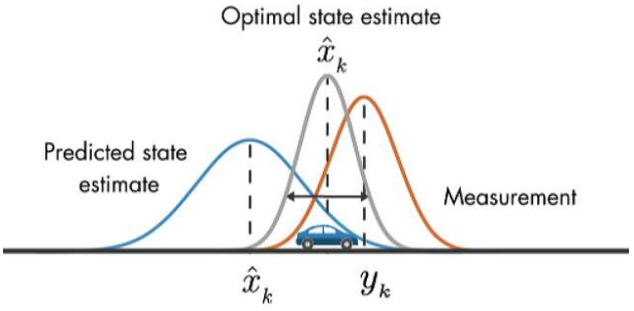
- 현재 상태 예측: 이전의 상태 정보를 바탕으로 현재 상태를 예측합니다. 노이즈가 존재하기 때문에, 일정한 불확실성을 가집니다.
- 불확실성 추정: 예측한 상태에 대한 불확실성을 추정합니다. 이전 상태의 불확실성뿐만 아니라, 외부 요인에 의한 노이즈를 반영합니다.

-업데이트 단계

- 새로운 데이터를 수집: 새로운 측정값을 얻습니다.
- 예측과 새로운 측정값 비교: 예측 단계에서 얻은 값과 새로운 측정 값을 비교합니다.
- Kalman filter는 예측과 새로운 측정 값의 신뢰도를 조정합니다. 즉, 새로운 정보를 얼마나 신뢰할 것인지를 계산하여 다음 추정량에 반영합니다.



이와 같이 Kalman Filter는 추정값과 업데이트된 값의 차이를 반영하여 filtering해주기 때문에 노이즈 처리에 용이합니다.

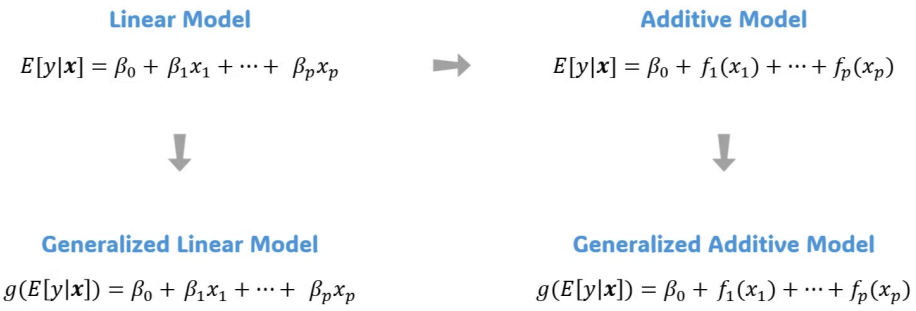


정리하면, 칼만 필터는 잡음이 포함된 과거 측정값에서 현재 상태의 결합분포를 추정하는 알고리즘입니다. 어떠한 정보가 있을 때 해당 데이터는 mixture 모델임을, 즉 다른 분포들의 결합이라는 것을 가정합니다. 일반화된 분포를 가정하는 것이 아닌 데이터의 특성에 맞는 분포 모델링을 할 수 있다는 것이 특징입니다.

II. Prophet 모형

1. 개념

Prophet 모델은 GAM(General Additive Models) 알고리즘의 일종으로, additive model을 기반으로 선형적인 트렌드에 seasonality나 holiday effect를 더해서 만들어집니다. 이때 GAM이란, 일종의 회귀와 유사하지만 x가 individual predictor가 아닌 smooth function이라고 생각하시면 됩니다. 단순한 function들을 더해서 더 복잡한 패턴을 만들게 됩니다.



Prophet은 시계열을 추세, 계절성, holiday 세 가지의 smooth function을 표현한 것입니다.

Prophet 수식

$$y_t = g_t + s_t + h_t + \epsilon_t$$

$g(t)$: linear or logistic curve

$s(t)$: periodic changes

$h(t)$: holiday effects

$e(t)$: error term

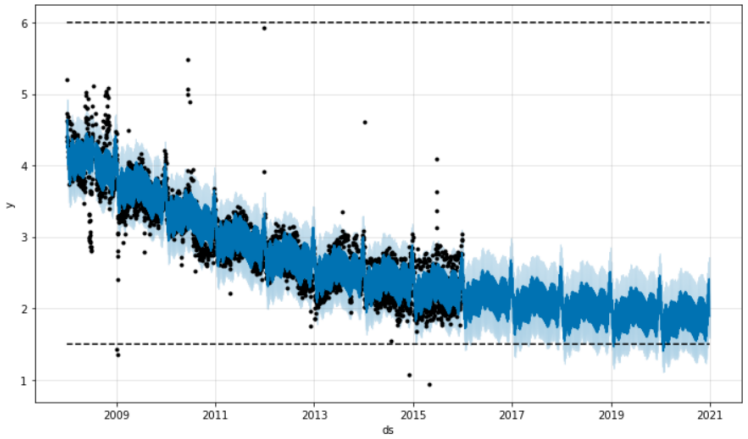
직관적으로 설명하면, 데이터를 보고 추세를 그린 뒤, 데이터를 보고 계절성을 그립니다. 또 다시 데이터를 보고 holiday의 영향력을 그립니다. 이 그린 것들을 모두 합쳐서 만든 게 prophet입니다.

이때 holiday는 '특징적인 사건'을 의미합니다. 말그대로 휴일을 의미하기보다 특정 구간 안에서 무작위적으로 여러 번 일어나는 경우라고 생각하시면 됩니다. 예컨대 '크리스마스마다 모임의 수가 증가해서 레스토랑의 매출이 증가한다'는 패턴은 계절성으로도 충분히 잡아낼 수 있습니다. 오히려 '축구경기가 있을 때마다 치킨집 매출이 상승한다'와 같이 랜덤한 사건에 대한 term이라고 생각하시면 되겠습니다.

2. Prophet 모형의 구성요소

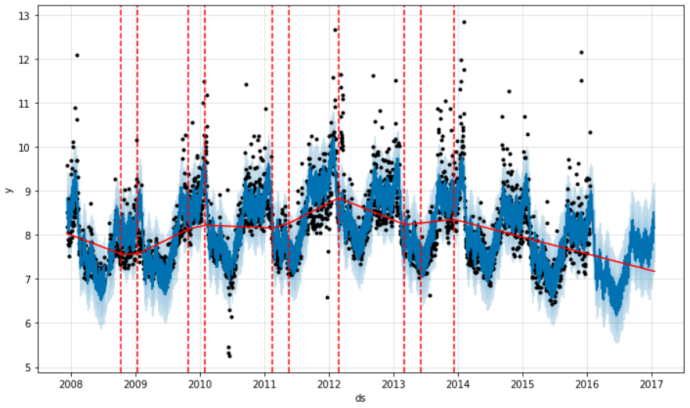
i. **추세**

Prophet 모델은 두 가지 방식으로 추세를 탐지합니다. 첫 번째로, 아래의 그래프처럼 예측하고자 하는 값의 한계(상한 or 하한)가 존재하고, nonlinear한 경우 우리는 이를 반영한 모델을 세워야 합니다. 이를
saturating growth model 이라고 합니다.



$$g(t) = \frac{C}{1 + exp(-k(t-m))}$$

두 번째로, 변곡점(changepoint) 분석을 통하여 추세를 추정할 수 있습니다. 프로펫에서는 이를 **linear trend with changepoints** 한 경우로 칭합니다.



linear trend에서의 turning point 계산에 대해 알아보도록 하겠습니다. 시계열 자료에서 변곡점을 기점으로 상승세나 하락세의 흐름이 변화합니다. Prophet 모델은 다음과 같은 식을 통하여 이러한 변곡점을 자동으로 감지합니다.

$$g(t) = (K + \alpha(t)^T\delta)t + (m + a(t^T)\gamma)$$

이 식은 g(t) = ( )t + ( )의 형태인데요, k는 처음의 기울기, $a(t)$는 특정 turning point보다 크면 1이고 아닌 경우 0인 벡터입니다. δ는 (직전 기울기 - 해당 turning point 이후의 기울기)를 뜻합니다.절편을 보도록 하겠습니다. m은 처음의 절편이고, γ는 직선들이 연속이도록 하는 절편의 변화입니다.

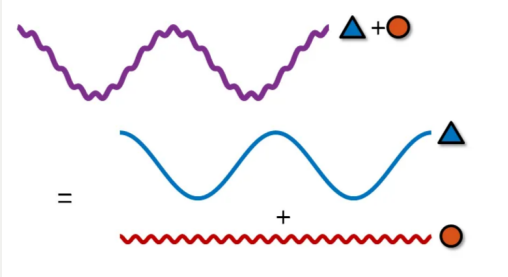
ii. **계절성**

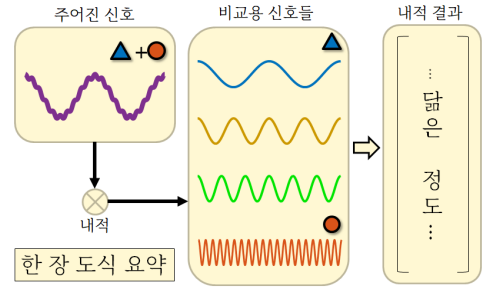
Prophet 모형의 seasonality는 푸리에 급수(Fourier Seires)에 근간을 두고 있습니다. 이는 주기함수를 삼각함수의 가중치로 분해한 급수입니다. 시계열을 주파수 영역에서 분석하는 과정에서 보통 시계열을 푸리에 급수로 표현하여 분석합니다.

$$s(t) = \sum_{n=1}^{N}(a_n cos(\frac{2\pi nt}{P}) + b_n sin(\frac{2\pi nt}{P}))$$

이때 a_n, b_n 계산은 자동적으로 모델에 의해 완성되며, N을 지정해주어야 합니다. N은 각 주기함수(sin, cos)가 몇 개씩 들어갈지에 대한 파라미터로, N이 커질 경우 seasonlity가 잘 반영되지만 과적합의 위험이 존재합니다. 2주차에서 배운 AIC와 같은 모델 선택 과정을 통하여 상세히 조절할 수 있습니다.

푸리에 급수





푸리에 급수(혹은 변환)이란 간단하게 말해 여러 주파수 성분을 담고 있는 신호를 각 주파수 성분별로 분해해주는 역할을 하는 것입니다. 푸리에 급수를 수행할 때는 분석하고자 하는 주기 신호의 가장 큰 주기(혹은 가장 낮은 주파수)를 확인하고 그 주파수의 정수배가 되는 주파수 미리 준비해둡니다. 이후 '내적(inner product)'을 이용해 각 주파수들과 닮은 정도를 계산하여 각 주파수 성분이 얼마만큼 들어있는지를 검사하는 과정인 것이다.

iii. **Holiday Effect (Event)**

불규칙 변동에 대한 분석을 진행합니다. 예를 들어 추수감사절의 경우, 11월의 네 번째 목요일이기 때문에 매년 날짜가 조금씩은 달라집니다.

$$h(t) = Z(t)_k,$$
$$where\ Z(t) = [1(t \in D_1), \dots, 1(t \in D_L)]$$

이때 D_i는 i번째 holiday의 날짜들이 들어 있는 벡터입니다. Z(t)는 holiday에 해당할 경우 1, 아닐 경우 0을 반환합니다. k는 가중치입니다.

3. Prophet 모델 적합 과정

Prophet 모형은 backfitting algorithm 혹은 L-BFGS 방식을 통해 모델을 적합합니다. 보통 후자를 더 자주 사용하기 때문에 L-BFGS 방식에 대해 알아보도록 하겠습니다.

BFGS는 알고리즘 최적화 방법입니다. 알고리즘을 최적화할 때 우리는 1차 미분을 사용하는 방법과 2차 미분을 사용하는 방법 중 하나를 택하게 됩니다. 이때 1차 미분으로 구하는 방법의 대표적인 예시로 gradient descent가 있습니다. 다만 해당 방법은 learning rate에 따라서 local minumum으로 가는 문제가 발생할 수 있으며, 2차 미분이 더 빠르기 때문에 2차 미분을 더 많이 사용합니다. 다만, hessian matrix(2차 미분한 행렬)을 구하는 것은 매우 어렵거나 불가능한 경우가 많아 미분값의 근사값을 구하는 다양한 방법론이 발달하였습니다. 그중 가장 빠른 편에 속하는 방법이 L-BFGS입니다.

BFGS 방법은 위에서 언급된 2차 미분을 통해 근삿값을 찾는 방법인 Newton's method($f''(x_i)(x_{i+1} - x_i) = -f'(x_i)$)와 유사한 형태를 지니고 있습니다. x를 vector form이라고 할 때, BFGS는 다음과 같습니다.

$$B_i p_i = -\nabla f(x_i)$$

이때 B_i는 ith stage에서의 Hessian matrix에 대한 approximation입니다. p_i는 $x_i x_i x_{i+1}$를 찾는 방향(search direction)을 의미합니다.

알고리즘 구현 방식에 대해 알아보겠습니다.

i. initial x_0와 H_0를 설정합니다. 보통 H_0는 단위행렬로 설정됩니다.

ii. $B_i p_i = -\nabla f(x_i)$를 풀어서 p_i를 계산합니다.

iii. $x_{i+1} = x_i + \alpha_i p_i$: Line Search를 진행하여 α_i를 구하고, 이를 반영하여 x_{i+1}을 계산합니다.

iv. 새로운 점 x_{i+1}을 계산한 후, 다음과 같이 기울기 변화를 업데이트합니다. $y_i = \nabla f(x_i + 1) - \nabla f(x_i)$

이 과정을 x_i로 수렴할 때까지 반복합니다.

L-BFGS란 BFGS 알고리즘의 변형으로, 고차원 문제에서 **메모리 효율성을 개선**한 방식입니다. BFGS는 Hessian 행렬을 직접 계산하지 않고 근사값을 사용하여 2차 미분 정보를 추정하지만, 고차원 문제에서는 메모리와 연산 비용이 크게 증가할 수 있습니다. L-BFGS는 이를 해결하기 위해 Hessian 행렬 전체를 저장하는 대신, **최근 업데이트된 값들만을 이용**하여 Hessian의 근사값을 계산합니다.

Prophet 모형은 위와 같은 L-BFGS의 원리를 통하여 한 TS의 추정값을 구합니다. 휴~

4. Prophet 모델의 장점 및 단점

i. 장점

- 모델링을 할 때 매우 정확하고 빠르다.
- seasonality에 대한 사전정보가 있을 때 쓰기에 적합하다.
- daily data일 때, 혹은 데이터가 상대적으로 작을 때 사용하기 유리하다.
- 알아서 결측치와 이상치를 처리해준다.

ii. 단점

- 여러 개의 target 변수가 있을 때 (multivariate) 연산이 오래 걸리며, 해당 변수들 간의 관계를 고려하지 않는다.
- seasonality 혹은 holiday 이상의 성질들이 존재할 경우 반영하기 어렵다.

→ 이러한 단점을 해결하기 위해 다른 알고리즘과 합쳐서 더 좋은 모델을 만들 수 있습니다.

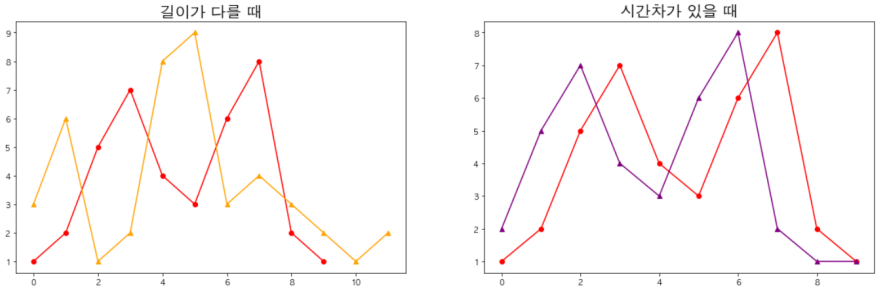
III. 시계열과 머신러닝

1. 클러스터링

클러스터링이란 서로 다른 성질을 가진 구성요소들로 이루어진 집단에 대해, 비슷한 패턴을 보이는 공통적인 특성을 지닌 구성요소들끼리 소집단(cluster)을 이루게 하여, 전체 집단을 이러한 소집단을 중심으로 분류하는 비지도 학습의 하나입니다. 즉 유사한 성격의 집단을 가까이 위치하고, 이질적인 집단은 멀리 위치하여 분석결과를 도출하는 방식입니다.

군집분석에도 여러 방법이 있습니다. 자료의 거리 또는 자료의 상관성을 토대로 한 유사도를 이용하여 dendrogram 형태의 군집형성을 수행하여 군집을 결정하는 계층적 군집분석 방법, 어떠한 개체가 어떠한 군집에 속하는가를 나타내는 비계층적 군집방법, 그리고 분석 모형에 의하여 군집을 결정하는 모형 군집분석 방법 등으로 구분됩니다.

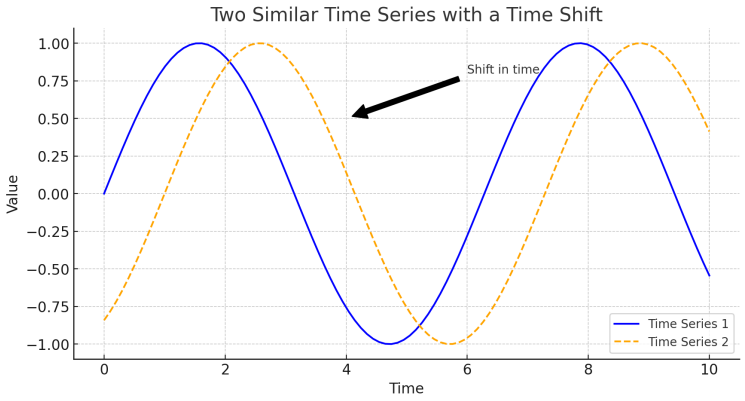
앞서 소개한 것과 같이 클러스터링 분석 시에는 거리 지표를 선택해야 합니다. 일반적인 거리 기반 클러스터링에서 사용되는 Euclidean 거리의 경우, 다음과 같은 문제점이 발생할 수 있습니다.



왼쪽 그림과 같이 길이가 다른 경우, Euclidean 거리를 계산할 수 없습니다. $X = (x_1, x_2, \ldots, x_k)$, $Y = (y_1, y_2, \ldots, y_k)$의 거리는 $\sqrt{\sum_{j=1}^{k}(x_i - y_i)^2}$로 계산되는데, 시계열의 길이가 다르기 때문에 해당 식으로는 거리를 구할 수 없는 것이죠. 오른쪽 그림의 경우, 보라색 시계열이 빨간색 시계열에 선행되지만, 단순히 시간차로 인하여 Euclidean 거리가 크게 계산되어 다른 시계열로 분류하게 됩니다. 이러한 문제점을 해결하며 시계열 데이터는 시간을 인식하는 거리 측정법을 사용해야 하는데, 그중 가장 잘 알려진 DTW에 대해 알아보겠습니다

2. DTW

동적시간왜곡 DTW(Dynamic Time Wraping)는 시간, 속도 또는 길이가 정확히 정렬되지 않은 두 시계열 간의 유사성을 측정하는 방법입니다. 이 알고리즘은 음성 인식에서 처음 나온 방법으로 패턴 인식에서 이용되었고 시퀀스를 시간의 길이를 고려하지 않고 인식할 수 있는 방법입니다. 음성의 경우에는 두 음성의 빠르기를 생각하지 않고 패턴의 진행, 두 음성의 모양을 비교하는 데 사용됩니다.



일반적으로 두 시계열 데이터를 비교하는 방법은 동일 시점에서 두 데이터의 값을 비교하는 것입니다. 하지만, 비슷한 패턴을 가진 두 시계열 데이터도 같은 시점에서 비교했을 때 큰 차이가 날 수 있습니다. 특정 시점에서의 두 데이터의 값이 중요한 분석일 때는 이러한 방식이 맞습니다. 하지만, 두 시계열 데이터 간의 패턴 유사도가 중요한 분석에서는 다른 방법이 필요합니다.

DTW는 한 시계열 데이터의 첫 시점부터 순차적으로 다른 시계열 데이터 내에서 가장 비슷한 시점을 찾아, 이 최소 거리들을 누적하여 유사도를 측정합니다. 이로 인해, 길이와 시점의 차이가 있는 시계열 데이터도 유사도를 비교할 수 있습니다.

두 시계열 X와 Y가 있을 때, X이 각 요소와 Y의 가장 가까운 점 사이의 거리 제곱 합계의 제곱근으로 계산됩니다. 아래의 식을 통해 자세히 알아보겠습니다.

$$X = (x_0, \ldots, x_n), Y_n = (y_0, \ldots, y_m)$$
$$DTW(x, y) = min_\pi \sqrt{\Sigma_{(i,j)\in\pi} d(x_i, y_i)^2}$$

이때 π는 다음과 같은 조건을 만족합니다.

- $\pi_k = (i_k, j_k)\ with\ 0 \le i_k < n\ and\ 0 \le j_k < m$
- $\pi_0 = (0,0)\ and\ \pi_{K-1} = (n-1, m-1)$
- $for\ all\ k > 0, \pi_k = (i_k, j_k)\ is\ related\ to\ \pi_{K-1} = (i_{K-1}, j_{K-1})\ as\ follows:$
  - $i_{k-1} \le i_k \le i_{k-1} + 1$
  - $j_{k-1} \le j_k \le j_{k-1} + 1$

위 식이 어떤 과정으로 이루어지는지 더 구체적으로 알아보겠습니다.

$$X = (1,2,3,3,2,1),\ Y = (1,1,2,3,3)$$

X/Y	1	1	2	3	3
1	|1-1|=0	0+|1-1|=0	0+|1-2|=1	3	5
2	0+|1-2|=1	0+|1-2|=1	0+|2-2|=0	1	2
3	1+|1-3|=3	3	1	0	0
3	3+|1-3|=5	5	2	0	0
2	5+|1-2|=6	6	2	1	1
1	6+|1-1|=6	7	3	3	3

i. 두 벡터의 길이를 행, 열로 가지는 행렬을 그립니다.

ii. 첫 번째 열과 첫 번째 행은 각각 첫 번째 X와 Y, 첫 번째 Y와 X의 유클리디안 거리를 누적합으로 구합니다.

iii. i, j 번째 값을 구할 때 아래의 예시와 같이 구합니다.

ex) 현재 distance = d = |2-1| = 1

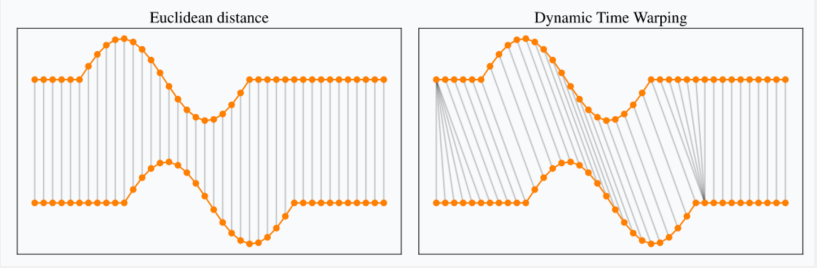
$$min(d_{i-1}+d, d_{i,j-1}+d, d_{i-1,j-1}+2d) = min(0+1, 1+1, 0+2) = 1$$

P \ Q	1	7	3	4	1	10	5	4	7	4
4	42	24	-	-	-	-	-	-	-	-
8	39	21	-	-	-	-	-	-	-	-
6	32	20	-	-	-	-	-	-	-	-
2	27	19	-	-	-	-	-	-	-	-
3	26	14	-	-	-	-	-	-	-	-
9	24	10	-	-	-	-	-	-	-	-
10	16	8	12	**11**	-	-	-	-	-	-
5	7	5	5	5	-	-	-	-	-	-
4	**3**	3	4	4	7	13	14	14	17	17
1	0	6	**8**	11	11	20	24	27	33	36

|10-4|+min(주변애들) = 6+5 =11

DTW에서 한 시계열의 모든 시간은 최소한 다른 시계열의 한 시간에 대응해야 합니다. 또한, 각 시계열의 처음과 끝은 서로 처음과 끝에 대응해야 합니다. 마지막으로 시간 간의 매핑은 과거가 아니라 미래로 이동하는 관계만 표현해야 합니다. 한 시계열에서 시간축 상 이미 지난 시간을 다른 시계열의 시간에 대응할 수는 없는 것이죠.

Euclidean Distance vs DTW

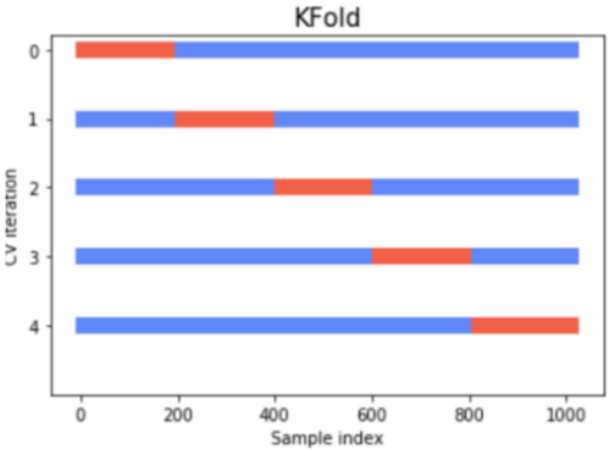


두 거리를 이용한 유사도는 모두 매칭된 데이터들 간의 거리의 합이라는 점에서는 비슷합니다.
유클리드 거리의 경우 계산이 용이하고 연산속도가 뛰어나다는 장점이 있지만 패턴의 떨림과 움직임이 심해질수록 결과가 어긋나는 현상이 발생합니다. 또한 길이가 다른 시계열은 분석이 불가하다는 단점이 있습니다. 사실 앞서 소개한 K-means 클러스터링의 경우 유클리드와 비슷한 거리가 더 적절한 거리지표입니다. 따라서 거리 지표를 선택할 땐 해당 지표가 분석 방법의 가정, 강점 및 약점을 반영하는지 확인한 후 적절한 지표를 선택해야 합니다.

항상 DTW 가 좋은 결과를 도출해주는 것은 아닙니다. DTW 의 왜곡의 정도에 아무런 제약을 주지 않는 경우에는 너무 과도하게 왜곡되어 유사하지 않은 시계열들을 유사한 것처럼 비교할 가능성이 존재하기 때문입니다. 따라서 왜곡할 수 있는 범위를 제약하는 contrained DTW를 사용하기도 합니다.

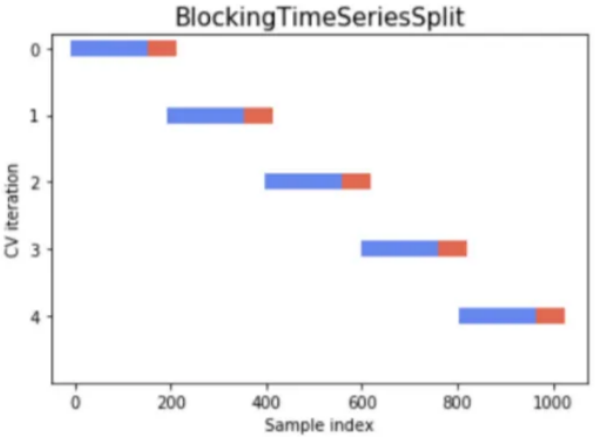
3. 교차검증

교차검증(Cross Validation; CV)은 과적합을 방지하기 위한 방법 중 하나로 신뢰성 있는 모델 평가를 진행하기 위해 필요합니다. 모델을 검증하기 위해 전체 데이터를 train set과 validation set으로 나누어 평가하는 것이 바로 CV입니다.



시계열자료분석에서도 이러한 CV 과정은 필요하나, 일반적으로 사용하는 K-fold CV 등을 사용할 수 없다는 문제점이 존재합니다. 그 이유는 시계열이라는 것은 말 그대로 시간의 순서가 굉장히 중요한 자료인데, 일반적인 CV 과정은 위 그림과 같이 시간 순서를 고려하지 않기 때문입니다!
시계열 데이터를 위한 CV 기법이 따로 존재하는데요! 시계열 교차검증 기법 중 blocked time series cv와 일반적인 time series cv에 대해 알아보겠습니다.

i. **Blocked Time Series CV**

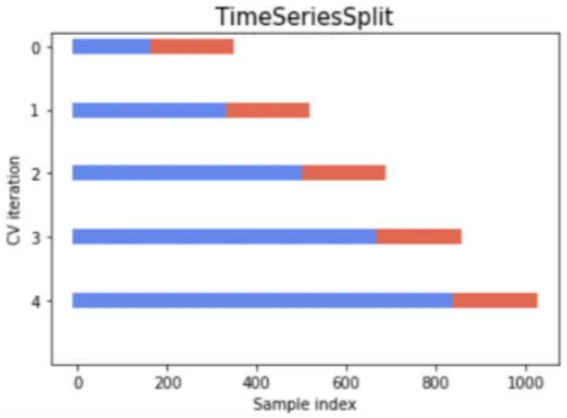


Rolling window CV라고도 불리는 방법입니다. Rolling window란 동일한 사이즈의 window를 옆으로 이동시킨다는 의미입니다. 같은 사이즈의 window내에서 일정 비율로 train과 validation을 분할해 교차검증을 진행합니다.

🥦 Dataset: [1,2,3,4,5] → 아래의 규칙에 따라 train/test set을 만들려고 합니다.
- 모든 train:test의 비율은 2:1이다.
- train+test의 사이즈는 동일하게 유지된다.
다음과 같이 데이터셋을 형성하는 방식이 blocked time series CV입니다.
train: [1,2] test: [3]
train: [2,3] test: [4]
train: [3,4] test: [5]

이 이후에는 다른 CV과정과 동일하게 각 데이터 셋에서 구한 정확도(오차)의 평균을 구하여 모델의 성능을 확인합니다.

ii. **Time Series CV**



Expanding window forecast 라고도 합니다. Expanding window 는 누적하며 이동한다는 의미입니다. 위 그림과 같이 데이터가 시간 순으로 정렬되어 있을 때 가장 먼저 제일 작은 사이즈의 train set 을 이용하여 test set을 예측하고, 그 다음 이전 단계에서 train, test에 활용되었던 데이터를 전부 train set으로 다시 활용하는 과정을 반복하면서 점차 train set 의 크기를 늘리는 방법입니다. 아래의 예시를 통해 알아보겠습니다.

Dataset: [1,2,3,4,5] → 아래의 규칙에 따라 train/test set을 만들려고 합니다.
- 모든 test set은 unique하다.
- trai set의 관측치는 직전 단계에서 test로 활용된다
다음과 같이 데이터셋을 형성하는 방식이 blocked time series CV입니다.
train: [1] test: [2]
train: [1,2] test: [3]
train: [1,2,3] test: [4]

trian: [1,2,3,4] test: [5]

이 이후에는 다른 CV과정과 동일하게 각 데이터 셋에서 구한 정확도(오차)의 평균을 구하여 모델의 성능을 확인합니다.

4. 클래스 불균형

모델링을 진행하다보면 클래스 불균형을 자주 만나볼 수 있습니다. 일반적인 데이터에서는 샘플링을 통해 클래스 불균형을 해결하지만, 시계열 데이터에서는 마찬가지로 시간의 흐름을 반영해야 하기 때문에 샘플링을 사용하기 어렵습니다. 따라서 시계열 데이터에서는 더 적은 수를 가지는 클래스에 가중치를 부여하는 방법으로 클래스 불균형을 해결해야 합니다.

클래스 불균형 해결방법

i. Scale_pos_weight
이진분류 문제에서 사용하는 파라미터로, 기본 설정은 0과 1로 라벨링되었음을 가정. 수가 더 적은 쪽을 1로 섦정하며, 해당 클래스에 가중치를 부여하는 방식

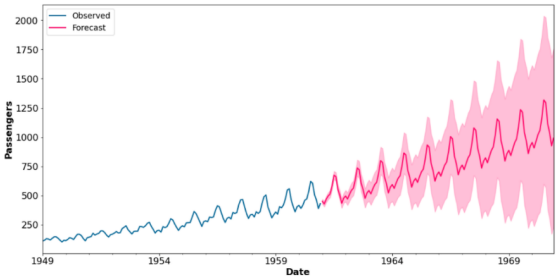
ii. Class_weight
수가 더 적은 쪽에 가중치를 부여하는 방법. model.fit() 또는 model.fit_generator() 파라미터로 사용하여 클래스별 가중치를 직접 작성
ex) model.fit(class_weight = {0:1, 1:2})

iii. Sample_weight
다중 분류에서 사용하며, 각 클래스 비율의 역수를 가중치로 계산하는 함수

ex)class_weight.compute_sample_weight(class_weight = 'balanced')

5. 예측 및 평가지표

전처리를 하고 어떤 모형을 사용할지 정했다면, 미래 값들에 대해 예측하고 해당 모형이 예측을 얼마나 잘 하는지 살펴봐야겠죠? 학습된 모델을 사용하여 미래 값을 예측하는데 예측 구간과 신뢰 구간을 고려하여 결과를 해석하는 것이 중요합니다.



시계열 모델링에서 주로 사용되는 평가지표에는 MAPE와 SMAPE가 있습니다.

i. **MAPE** (Mean Absolute Percentage Error)

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}|\frac{y_t - \hat{y}_t}{y_t}|$$

MAPE는 MAE($\frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|$)를 비율로 나타낸 것으로, 실제값과 예측값 사이의 차이를 실제값으로 나눠줌으로써 오차가 실제값에서 차지하는 상대적인 비율을 산출합니다. 그리고 해당 값을 절대값 취한 후 평균을 구합니다. 오차의 정도를 백분율 값으로 나타내기 때문에 모델의 성능을 직관적으로 이해하기 쉬우며, 타겟 변수가 여러개 일 때 각 변수별 모델의 성능을 평가하기 용이합니다.

하지만 실제값에 0이 존재한다면 MAPE가 정의 되지 않는 문제점이 있습니다. 또한 절대적인 값의 오차가 같더라도 실제값과 예측값과의 대소 관계에 따라 과대 추정하는 예측값에 패널티를 더 부여합니다.

```
평균 오차가 20일 때 실제값과 예측값의 대소 관계에 따른 MAE, MAPE 비교

실제값이 예측값 보다 작을 때 (예측값이 과대추정)
MAE: 20.0
MAPE: 0.9133333333333333

실제값이 예측값 보다 클 때 (예측값이 과소추정)
MAE: 20.0
MAPE: 0.4371428571428571
```

ii. **SMAPE** (Symmetric Mean Absolute Percentage Error)

$$SMAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|}$$

SMAPE는 MAPE가 지닌 한계점을 보완하기 위해 고안되었습니다. 위에서 MAPE는 0.91, 0.43의 다른 값을 산출했지만 SMAPE는 같은 값을 산출하는 것을 확인할 수 있습니다.

```
평균 오차가 20일 때 과소추정, 과대추정에 따른 MAE, SMAPE 비교

과대추정 시
MAE: 20.0
SMAPE: 0.14912698412698414

과소추정 시
```

```
MAE: 20.0
SMAPE: 0.21857142857142856
```

IV. 시계열과 딥러닝

딥러닝은 비선형성과 복잡한 패턴을 학습하는 데 강점을 가지며, 특히 방대한 데이터와 복잡한 구조의 시계열 데이터를 다루는 데 적합합니다. 딥러닝을 활용한 시계열 분석은 패턴 인식과 장기 종속성(Long-Term Dependency)을 효과적으로 학습할 수 있는 특징 덕분에 주목받고 있습니다.

시계열 데이터는 고유한 시간 의존성 구조를 가지고 있으며, 딥러닝은 이러한 특성을 학습하여 미래를 예측하는 데 강점을 보입니다. 딥러닝 모델들은 데이터 내의 비선형 패턴과 복잡한 의존 관계를 학습하여 예측 정확도를 높일 수 있습니다.

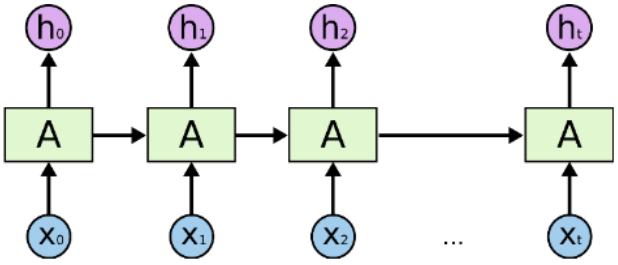
전통적 시계열 모델은 데이터의 시간적 관계를 명시적으로 정의하지만, 딥러닝 모델은 데이터에서 패턴을 자동으로 학습하여 이러한 관계를 내재적으로 이해합니다. 예를 들어, RNN과 LSTM은 과거 데이터의 패턴을 학습하여 미래를 예측하는 데 주로 사용됩니다.

1. RNN

Recurrent Neural Networks (RNN)는 시계열 데이터의 특성을 반영하기 위해 설계된 딥러닝 모델로, 과거 데이터의 정보를 현재와 미래의 예측에 반영할 수 있는 구조를 가지고 있습니다. RNN은 시계열 데이터를 순차적으로 처리하면서 과거의 데이터를 기억하고 이를 바탕으로 미래의 값을 예측할 수 있습니다. 예를 들어, 주식 시장의 가격 예측에서 RNN은 이전 시간의 주가 변동을 기반으로 다음 시간의 주가를 예측할 수 있습니다. 이는 전통적인 통계적 시계열 모델(예: ARIMA)이 과거 몇 개의 값만을 사용하여 선형적인 관계를 모델링하는 것과는 대조적입니다.

하지만, RNN은 긴 시퀀스의 시계열 데이터를 처리할 때 **기울기 소실 문제**로 인해 장기적인 의존성을 학습하기 어렵다는 한계가 있습니다. 예를 들어, 기후 데이터의 장기 패턴을 예측할 때 RNN은 데이터를 제대로 반영하지 못할 수 있습니다. 이 때문에 RNN은 상대적으로 짧은 시간 범위에서의 예측에 더 적합하며, 주로 단기 예측이나 비교적 짧은 기간의 패턴 분석에 사용됩니다. 이를 개선하기 위해, LSTM과 같은 변형 모델이 개발되었습니다.

2. LSTM



Long Short-Term Memory (LSTM)는 RNN의 구조적 한계를 극복하기 위해 고안된 모델로, 시계열 데이터의 장기적인 의존성을 학습하는 데 매우 효과적입니다. 시계열 분석에서는 미래 값을 예측하기 위해 장기간의 과거 데이터를 고려해야 할 때가 많습니다. LSTM은 이를 가능하게 하는 모델입니다. 예를 들어, 에너지 소비 예측 문제에서, LSTM은 수년간의 계절적 패턴과 일일 변동성을 모두 학습하여 미래의 에너지 수요를 예측할 수 있습니다. 이는 전통적인 시계열 모델들이 계절적 효과와 추세를 따로 모델링하는 것과 달리, LSTM은 단일 모델 내에서 이러한 복잡한 패턴을 모두 학습할 수 있습니다.

LSTM은 **메모리 셀**과 **게이트 메커니즘**을 통해 중요하지 않은 정보를 잊고, 중요한 정보를 장기간 유지하면서 예측 성능을 높입니다. 예를 들어, 금융 시계열 데이터에서 LSTM은 과거의 주요 사건(예: 경제 위기)과 같은 중요한 정보는 기억하고, 일상적인 변동은 무시하면서 보다 정확한 장기 예측을 수행할 수 있습니다. 이러한 특성 덕분에 LSTM은 주식 가격 예측, 기후 데이터 분석, 장기 경제 예측 등 다양한 시계열 예측 작업에서 널리 활용됩니다.

LSTM은 시계열 데이터의 time dependency를 잘 반영할 수 있는 모델입니다. RNN 기반 모델은 이전의 학습 결과가 다음 학습 단계에 영향을 주기 때문에, 종속적인 데이터를 처리하는데 효과적입니다. 다만 randomwalk와 같이 시계열 데이터 간의 종속성이 불분명한 데이터에는 이러한 기법이 효과적으로 작동하지 않을 수 있습니다.

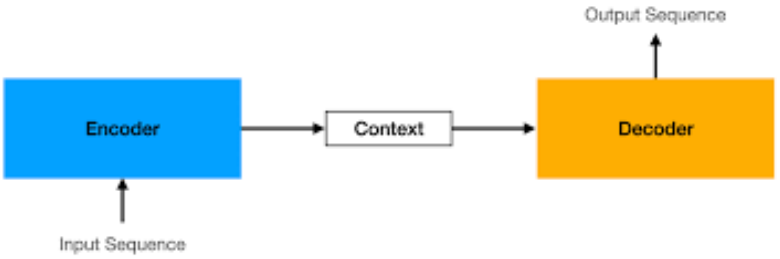
3. Transformer

i. Transfomer 구조

Transformer 모델은 시계열 분석에 새로운 패러다임을 제시한 모델로, **Self-Attention 메커니즘**을 사용하여 데이터를 순차적으로 처리하지 않고 병렬적으로 처리할 수 있습니다. 이는 특히 긴 시퀀스의 데이터를 다룰 때 효과적입니다. Transformer는 모든 타임스텝 간의 관계를 한 번에 고려할 수 있기 때문에, 시계열 데이터에서 장기적 패턴과 복잡한 상호작용을 학습하는 데 강력한 성능을 발휘합니다. 예를 들어, 주가 예측에 Transformer를 사용하면, 모델은 단순히 최근 데이터뿐만 아니라 수년 전의 데이터까지 한 번에 고려하여 더 정교한 예측을 할 수 있습니다. Transformer의 핵심은 RNN을 전혀 사용하지 않고, MLP 층으로만 구성된 multi-head attention으로 입력 시계열 간의 상관관계를 분석에 이용한다는 점입니다.

ii. 시계열 예측에서의 encoder - decoder 구조

Transformer를 이용하여 시계열 예측을 어떻게 진행하는지 이해하기 위해 transformer의 encoder, decoder 구조를 알아야 합니다.



일반적으로 자연어처리 task에서 사용되는 transformer는 encoder를 통해 입력받은 텍스트 데이터를 벡터화합니다. 최종적으로 가장 마지막에 출력되는 벡터를 디코더 부분으로 전달하는데, 해당 벡터에는 그 전 시점까지의 정보가 순차적으로 포함되어 있으므로 순서를 잘 반영할 수 있다고 보는 것입니다.

시계열 데이터에서도 마찬가지입니다. Encoder는 입력 데이터를 벡터화하여 고차원 표현으로 변환하는 역할을 수행합니다. Decoder는 이 벡터를 바탕으로 다음 시점의 값을 예측하거나 타겟 시계열 데이터를 재구성합니다.

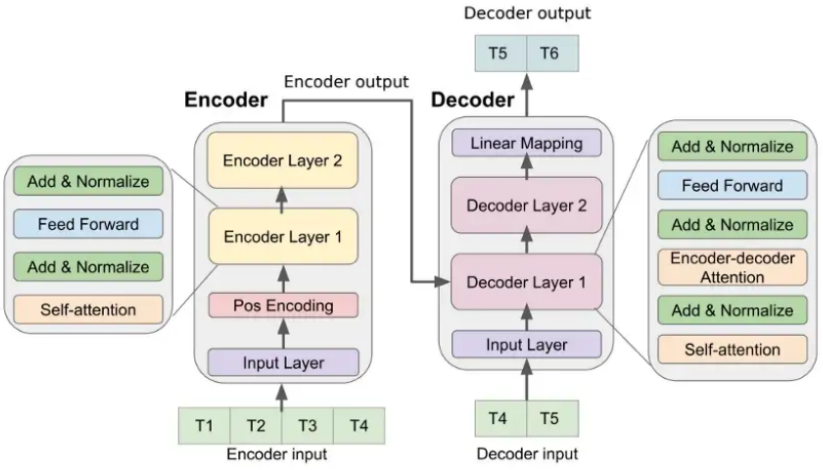


Figure 1. Architecture of Transformer-based forecasting model.

시계열 데이터에 대해 transformer를 사용하는 경우를 생각해봅시다. 연속된 4개의 시점의 시계열 (t_1, t_2, t_3, t_4)를 입력하여 (t_5, t_6)을 예측한다고 생각해봅시다. encoder에 입력되는 값은 (t_1, t_2, t_3, t_4)이 되고, decoder에는 출력 시계열보다 한 시점 앞선 (t_4, t_5)를 입력하여 decoder가 t_6를 예측하기 전 t_5를 예측하여 모델이 올바르지 않은 예측을 하지 않게 방지해줍니다. 즉, (t_4, t_5)를 한꺼번에 입력하지 않고 t_4를 통해 t_5를 우선 예측하도록 합니다. 그 이후 (t_4, t_5)를 입력하여 t_6을 예측하는 일종의 guide 역할을 하도록 자료를 차례로 축적하여 입력합니다. 이를 teacher forcing(교사 강요)이라고 합니다.

학습된 모형을 통해 미래치를 예측하는 경우를 좀더 구체적으로 살펴보도록 하겠습니다. 관측된 시계열 $y_1, y_2, \ldots, y_T$로 $y_{T+1}, y_{T+2}, \ldots, y_{T+h}$를 예측하는 경우를 고려해보겠습니다. Encoder input은 $(y_{T-k}, y_{T-k+1}, \ldots, y_T)$, decoder input은 $(0, 0, \ldots, 0)$을 입력하여 y_{T+1}을 예측합니다. 이 예측치를 $\hat{y}_{T+1}$라고 표현하겠습니다. 동일한 encoder input과 $(\hat{y}_{T+1}, 0, \ldots, 0)$을

decoder input으로 입력하여 $\hat{y}_{T+2}$를 구하고, decoder input을 $(\hat{y}_{T+1}, \hat{y}_{T+2}, \ldots, 0)$으로 수정한 후, y_{T+3}을 예측합니다. 이를 반복하여 decoder input을 $(\hat{y}_{T+1}, \hat{y}_{T+2}, \ldots, \hat{y}_{T+h-1}, 0)$으로 대체하여 y_{T+h}를 예측합니다. 그러므로 transformer에서는 decoder input을 **자기회귀적(autoregressive)** 방법으로 축적하여 미래값을 예측하게 됩니다.

iii. **transformer 모델의 장점**

Transformer 모델은 시계열 데이터의 **비정형적 패턴**을 잘 학습할 수 있는 장점을 가지고 있습니다. 기상 예측의 경우, Transformer는 갑작스러운 기후 변화나 드문 기상 패턴도 잘 학습하여 이를 반영한 예측을 수행할 수 있습니다. 또한, 병렬 처리 능력 덕분에 훈련 시간이 단축되고, 대규모 데이터셋을 효과적으로 사용할 수 있는 장점이 있습니다. 이는 빅데이터 환경에서의 시계열 예측 작업에 매우 유리합니다. 최근 Transformer는 특히 다변량 시계열 데이터 예측에서 우수한 성능을 보이며, 점점 더 많은 연구와 실무에서 채택되고 있습니다.

V. 혼합형 시계열 모형 -Neural Prophet

1. 정의

Neural Prophet은 전통적 통계 시계열 모형과 딥러닝 모형을 결합한 혼합(hybrid) 시계열 모형입니다. Neural Prophet은 전통적 통계 시계열 모형의 해석 용이성과 딥러닝 모형이가지고 있는 모형의 확장성과 정밀도 그리고 다변량 시계열 및 다중 시계열 데이터에 유연하게 적용할 수 있는 모형을 혼합한 시계열 모형입니다.

$$\hat{y}_t = T(t) + S(t) + A(t) + L(t) + F(t) + E(t)$$

T(t)	추세효과	L(t)	과거 공변량
S(t)	계절성	F(t)	미래 공변량
A(t)	자기회귀모형	E(t)	사건 및 공휴일 효과

이러한 Neural Prophet의 시계열 구성요소의 분해는 시계열 분석 결과의 설명 가능성(explainability)을 높여주고, 맞춤형 시계열 모형의 설계를 가능케 합니다.

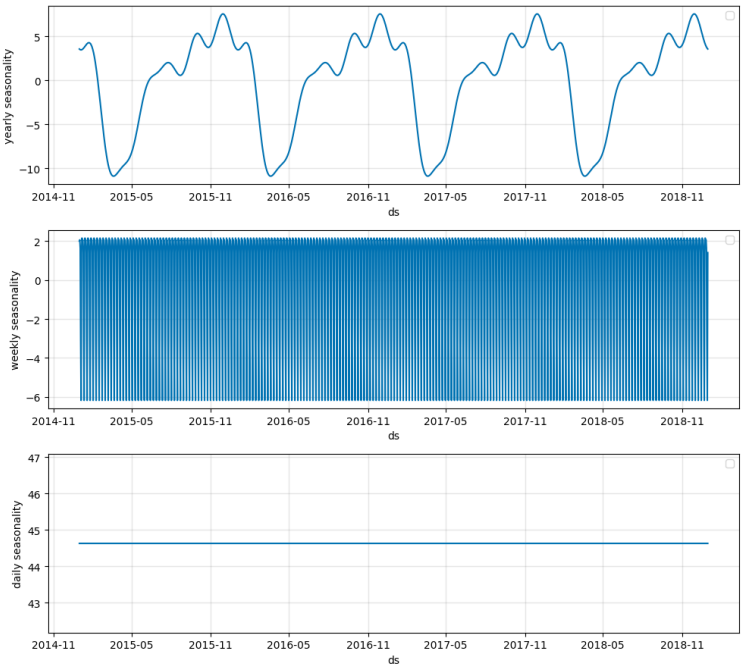
기존의 Prophet 모형은 단기 시점 예측을 위한 로컬 컨텍스트(local context)를 활용하지 못하였습니다. Nueral Prophet의 경우 위에서 언급한 자기 회귀($A(t)$) 및 공변량 ($L(t), F(t)$) 모듈로 해당 문제를 해결하였습니다. 또한 기존 Prophet 모형은 시계열의 예측 성능보다는 손쉬운 사용 및 예측 결과 해석에 초점을 맞춘 반면, Neural Prophet은 기존 모형의 우수한 해석력에 더하여 예측 성능을 개선했다는 점에서 의의를 지닙니다.

2. Neural Prophet 모형의 구성요소

각 구성요소들이 기존 모형과 비교했을 때 어떻게 발전되었는지 알아보도록 하겠습니다.

- Seasonality($S(t)$): 여러 개의 Fourier terms를 사용. 즉, 한 가지의 패턴만을 분석하는 것이 아니라 다중 계절성을 포착(weekly seasonality, monthly seasonality, yearly seasonality 등)

```
# Model and prediction
m = NeuralProphet(
    # Disable trend changepoints
    n_changepoints=0,
    # Enable all seasonality components
    yearly_seasonality=True,
    weekly_seasonality=True,
    daily_seasonality=True,
)
m.set_plotting_backend("plotly-static")
metrics = m.fit(df)
forecast = m.predict(df)
m.plot(forecast)
```



- Auto-Regressor($A(t)$): 기존의 classic AR 모형에서 전방향 신경망 네트워크를 적용한 AR-NET 활용

AR-NET

기존 classic-AR 모형의 computational 한계를 극복하기 위해 생성된 Neural Network 기반 AR 모형입니다. AR-Net의 장점은 $y_{t-p}, y_{t-p+1}, \ldots, y_{t-1}$을 입력하여 h개의 미래시점 예측치를 동시에 출력한다는 것입니다. (사실 이러한 장점은 AR-NET만 가진 장점이 아니라, 모든 머신러닝 및 딥러닝 시계열 모형이 가지고 있는 장점입니다.)

또한, AR-NET은 기여도가 작은 AR 모수는 0으로 수렴시키되, 중요한 AR 모수는 가급적 그대로 유지하는 규제화를 진행합니다.

$$R(\phi) = \frac{1}{p}\sum_{i=1}^{p}\frac{2}{1+exp(-c_1||\phi_i||^{c_2{}^{-1}})} - 1$$

- Lagged Regressor($L(t)$): 모델 해석에 적용되는 변수들 (공변량)
- Future Regressor($F(t)$): 미래에 이미 알 수 있는 변수들을 회귀 예측에 활용

Neural Prophet은 시계열의 각 구성요소가 모듈화되어 있어, 각 구성요소를 쉽게 확장할 수 있습니다.